RETI INFORMATICHE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

SPECIFICHE DI PROGETTO A.A. 2016/2017

Il progetto consiste nello sviluppo di un'applicazione client/server. Sia il server che il client dovranno essere mono-processo e sfrutteranno l'I/O multiplexing per gestire più canali di input simultaneamente.
L'applicazione da sviluppare è il gioco della **"Battaglia navale"** seguendo il paradigma peer2peer.
"Battaglia navale" è un gioco testa a testa, in cui ciascuno dei due giocatori deve affondare le navi avversarie indovinandone la posizione in una griglia bidimensionale. *Per semplicità, la griglia sarà grande 6x6 caselle e ogni giocatore ha a disposizione 7 navi ognuna grande solo una casella*.
All'inizio del gioco i due giocatori devono entrambi disporre le proprie navi, dopodiché devono cercare di indovinare la posizione delle navi dell'avversario. A turno, ciascun giocatore fa un tentativo indicando una casella. Dopo ogni tentativo, l'avversario gli fornisce il risultato del tentativo ("colpito" o "mancato"). Vince il giocatore che per primo riesce ad affondare tutte le navi avversarie.
Per sviluppare l'applicazione devono essere realizzati due programmi, **battle_server** per il lato server e **battle_client** per il lato client.
Il server avrà il compito di memorizzare il nome degli utenti connessi e le porte UDP sulle quali questi si metteranno in ascolto.
Lo scambio di informazioni tra client e server avverrà tramite socket TCP. Queste informazioni saranno solo informazioni di controllo che serviranno per implementare la comunicazione peer2peer. Lo scambio di messaggi tra i client avverrà tramite socket UDP.

LATO CLIENT

Il client deve essere avviato con la seguente sintassi:

```
./battle_client <host remoto> <porta>
```

dove:

- **<host remoto>** è l'indirizzo dell'host su cui è in esecuzione il server;
- **<porta>** è la porta su cui il server è in ascolto.

All'avvio, i comandi disponibili per l'utente devono essere:

- **!help**
- **!who**

- **!quit**
- **!connect username**

Il client deve stampare tutti gli eventuali errori che si possono verificare durante l'esecuzione.
All'avvio della connessione il client **deve** inserire il suo username e la porta di ascolto UDP. Un *esempio* di esecuzione è il seguente:

```
$ ./battle_client 127.0.0.1 1234

Connessione al server 127.0.0.1 (porta 1234) effettuata con successo

Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!who --> mostra l'elenco dei client connessi al server
!connect username --> avvia una partita con l'utente username
!quit --> disconnette il client dal server

Inserisci il tuo nome: client1
Inserisci la porta UDP di ascolto: 1025
```

Si gestisca il caso in cui il nome scelto sia già registrato presso il server.

Comandi base

!help: mostra l'elenco dei comandi disponibili.
Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!who --> mostra l'elenco dei client connessi al server
!connect username --> avvia una partita con l'utente username
!quit --> disconnette il client dal server
```

!who: mostra l'elenco dei client connessi e lo stato del client, cioè se è occupato o meno in una partita.
Il server mantiene una lista dei client connessi in ogni momento. La lista contiene gli username, l'indirizzo IP e la porta di ascolto UDP con cui i client si sono registrati al momento della connessione.
Esempio di esecuzione:

```
> !who
Client connessi al server:
   client1(libero)
```

```
client2(occupato)
client3(occupato)
client4(libero)
>
```

!connect username: il client avvia una partita con l'utente username.

Gli errori da gestire sono:

- username inesistente
- username già occupato in una partita
- errori a livello protocollare

Più in dettaglio il client farà richiesta al server (sempre tramite TCP) per sapere se esiste l'utente **username**. Se esiste e non è occupato il server manderà una richiesta al client **username** per sapere se è intenzionato ad accettare la partita con il client. Se la risposta è affermativa allora il server comunicherà al client l'indirizzo IP e porta di ascolto UDP del client **username**. Se negativa il server risponderà con uno specifico messaggio di errore. La concorrenza tra standard input e socket dovrà essere gestita sempre tramite la primitiva **select()**.

Esempio di esecuzione:

```
> !connect username

username ha accettato la partita!
Posiziona 7 caselle:
<casella1>
<casella2>
…
E' il tuo turno.
#!shot <casella>
username dice: mancato :(

E' il turno di username.
username spara in <casella>. Mancato :)
E' il tuo turno.
#!shot …
…
```

Possibile segnalazione di errore:

```
> !connect username

Impossibile connettersi a username: utente inesistente.
```

```
> !connect username

Impossibile connettersi a username: l'utente ha rifiutato la partita.
```

!quit: il client chiude il socket con il server, il socket UDP ed esce.
Il server stampa un messaggio che documenta la disconnessione del client. Il server, inoltre, dovrà gestire in maniera appropriata la disconnessione di un cliente.

```
> !quit

Client disconnesso correttamente
```

Comandi per il gioco

Quando la partita è avviata il client dovrà accettare i seguenti comandi:

- **!help**
- **!disconnect**
- **!shot square**
- **!show**

Se una partita è avviata si deve capire dal primo carattere della shell:

- **>** shell comandi, accetta i comandi base
- **#** shell partita, accetta i comandi relativi al gioco

!help: mostra l'elenco dei comandi disponibili.
Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:
!help --> mostra l'elenco dei comandi disponibili
!disconnect --> disconnette il client dall'attuale partita
!shot square --> fai un tentativo con la casella square
!show --> visualizza griglia di gioco
```

!disconnect: disconnette il client dall'attuale partita.

```
# !disconnect

Disconnessione avvenuta con successo: TI SEI ARRESO
```

Quando un client esegue una disconnessione comunicherà al server (tramite TCP) che l'utente è di nuovo libero. Il client con il quale era in corso la partita dovrà essere avvisato della disconnessione e dovrà mostrare un messaggio di vittoria.

`!show`: il client mostra due griglie, quella avversaria e quella del giocatore locale. In particolare quella avversaria mostrerà i tentativi effettuati finora dal giocatore locale con lo stato "colpito" o "mancato", quella locale mostrerà le proprie navi con lo stato "vivo" o "colpito" e, opzionalmente, i tentativi a vuoto dell'avversario.

LATO SERVER

Il server **battle_server** si occupa di gestire le richieste provenienti dai client: Tramite l'uso della **`select()`**, accetterà nuove connessioni TCP, registrerà nuovi utenti e gestirà le richieste dei vari client per aprire nuove partite.

La sintassi del comando è la seguente:

```
./battle_server <porta>
```

Dove **`<porta>`** è la porta su cui il server è in ascolto.

Una volta eseguito, **battle_server** deve stampare a video delle informazioni descrittive sullo stato del server (creazione del socket di ascolto, connessioni accettate, operazioni richieste dai client ecc.).

Un *esempio* di esecuzione del server è il seguente:

```
$ ./battle_server 1235
Indirizzo: 127.0.0.1 (Porta: 1235)

Connessione stabilita con il client
zerocool si e' connesso
zerocool è libero
Connessione stabilita con il client
acidburn si e' connesso
acidburn è libero
zerocool si è connesso a acidburn
acidburn si è disconnesso da zerocool
zerocool è libero
acidburn è libero
```

AVVERTENZE E SUGGERIMENTI

- Client e server si scambiano dei dati tramite socket TCP, i client si scambiano dati tra loro tramite UDP. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket. **Non è ammesso che vengano inviati su socket numeri arbitrari di byte**.
- Per memorizzare la griglia e per indirizzarla si utilizzi la struttura dati che si preferisce. In generale si lascia libertà di implementazione per ciò che non è esplicitamente specificato.
- Prevedere che il client si disconnetta in automatico da una partita dopo **1 minuto di inattività**, cioè nel caso in cui:
    - Non viene scritto niente nello standard input per un minuto
    - Non si riceve niente sul socket UDP per un minuto

VALUTAZIONE DEL PROGETTO

Il progetto viene valutato durante lo svolgimento dell'esame. Il tutto verrà fatto girare su una macchina con sistema operativo **Debian 8**. Si consiglia caldamente di testare il sorgente su una macchina Debian 8 prima di venire all'esame.
La valutazione prevede le seguenti fasi:

1. **Compilazione dei sorgenti**. Il client e il server vengono compilati attivando l'opzione **`-Wall`** che abilita la segnalazione di tutti i warning, e **non dovranno presentare warning o errori**. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, **interpretando i messaggi forniti dal compilatore.**
2. **Esecuzione dell'applicazione**. Il client e il server vengono eseguiti simulando una tipica sessione di utilizzo. In questa fase si verifica **il corretto funzionamento dell'applicazione e il rispetto delle specifiche fornite.**
3. **Esame del codice sorgente**. Il codice sorgente di client e server viene esaminato per controllarne l'implementazione. **Può essere richiesto di apportare modifiche al codice sul momento.**